PRIFYSGOL ABERYSTWYTH UNIVERSITY

Aberystwyth University

A Framework for the Scoring of Operators on the Search Space of Equivalence Classes of Bayesian Network Structures

Shen, Qiang; Daly, Ronan

Publication date:
2005

Citation for published version (APA):
Shen, Q., & Daly, R. (2005). *A Framework for the Scoring of Operators on the Search Space of Equivalence Classes of Bayesian Network Structures*. 67-74. http://hdl.handle.net/2160/435

General rights
Copyright and moral rights for the publications made accessible in the Aberystwyth Research Portal (the Institutional Repository) are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the Aberystwyth Research Portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the Aberystwyth Research Portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

tel: +44 1970 62 2400
email: is@aber.ac.uk

Download date: 04. May. 2023

A Framework for the Scoring of Operators on the Search Space of Equivalence Classes of Bayesian Network Structures

Rónán Daly
School of Informatics
University of Edinburgh
Edinburgh, EH8 9LE
Ronan.Daly@ed.ac.uk

Qiang Shen
Department of Computer Science
University of Wales, Aberystwyth
Aberystwyth, SY23 3DB
qqs@aber.ac.uk

Abstract

A method is proposed, whereby a particular application of an operator, applied to a structure representing a Bayesian network equivalence class can be scored in a generic fashion. This is achieved by representing a particular compound operator in terms of a finite set of primitive operators and finding the score of the compound operator through the influence of the primitive operators on the equivalence class. This method could be used in a Bayesian network structure learning framework which allows arbitrary definition of operators at runtime, by the composition of primitive operators.

1 Introduction

The task of learning Bayesian networks from data has, in a relatively short amount of time, become a mainstream application in the process of knowledge discovery and model building. The reasons for this are many.

For one, the model built by the process has an intuitive feel—this is because Bayesian networks consist of a directed acyclic graph (DAG), with conditional probability tables annotating each node. Each node in the graph represents a variable of interest in the problem domain and the arcs can (with some caveats) be seen to represent causal relations between these variables—the nature of these causal relations is governed by conditional probability tables associated with each node/variable. An example Bayesian network is shown at Figure 1.

Another reason for the usefulness of Bayesian networks is that aside from the visual attractiveness of the model, the underlying theory is quite well understood and has a solid foundation. A Bayesian network can be seen as a factorisation of a joint probability distribution, with the conditional probability distributions at each node making up the factors and the graph structure making up their method of combination. Because of this equivalence, the network

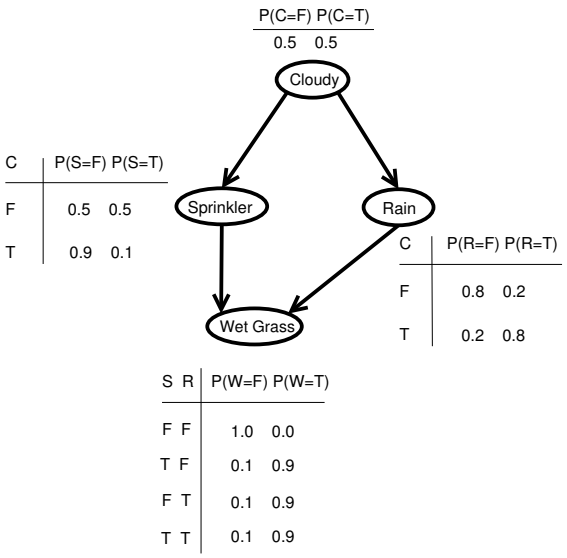


Figure 1: An example Bayesian network

can answer any probabilistic question put to it.

Finally, the popularity of Bayesian networks has been increased by the accessibility of methods to both query the model and learn both the structure and parameters of the network. It has been shown that inference in Bayesian networks is NP-Complete [5, 11], but approximate methods have been found to perform this operation in an acceptable amount of time. Learning the structure of Bayesian networks is also NP-Complete [1], but here too, heuristic methods have been found to render this operation tractable.

It is with the latter remark that this paper concerns itself, that is, the learning of the structure of a Bayesian network from a sample of data. There are generally two different methods used in this task. One uses statistical tests to uncover conditional independencies in the data and then uses these conditional independencies to produce the structure. The second defines a search on the space of Bayesian networks, using a scoring function defined by the implementor, which says how good the network is, rel-

ative to others.

This paper will seek to establish a framework to help with the scoring function mentioned in the second method in the previous paragraph. This framework will allow for the scoring of arbitrary changes in an equivalence class of Bayesian networks, with the minimum number of operations necessary. To this end, the rest of this paper will be structured in the following fashion.

Firstly, there will be a more in depth study of the problem of searching for an optimum Bayesian network, in both the space of Bayesian networks themselves and of equivalence classes of Bayesian networks. Then, a new method of scoring moves in the state space will be introduced, together with its motivation, benefits, consequences and implementation details. This algorithm will be analysed to prove its validity and to discover its complexity. Next, results of tests against other scoring methods will be discussed and finally, any conclusions and possible future directions will be stated.

2 Searching for a Bayesian Network Structure

To start out this section, some definitions and notation are introduced.

A graph $\mathcal{G}$ is given as a pair (V,E), where $V = \{v_1,\ldots,v_n\}$ is the set of vertices or nodes in the graph and E is the set of edges or arcs between the nodes in V. A directed graph is a graph where all the edges have an associated direction from one node to another. A directed acyclic graph or DAG, is a directed graph, without any cycles, i.e. it is not possible to return to a node in the graph by following the direction of the arcs.

A Bayesian network on a set of variables $V = \{v_1,\ldots,v_n\}$ is a pair $(\mathcal{G},\Theta)$, where $\mathcal{G} = (V,E)$ is a DAG and $\Theta = \{\theta_1,\ldots,\theta_n\}$ is a set of conditional probability distributions, where each θ_i is associated with each v_i.

In learning a Bayesian network from data, both the structure $\mathcal{G}$ and parameters Θ must be learned, normally separately. In the case of complete multinomial data, the problem of learning the parameters is easy, with a simple closed form formula for Θ [7]. However, in the case of learning the structure, no such formula exists and other methods are needed. As mentioned in the introduction, there are two main methods—using conditional independencies learned from the data and searching through a space of structures. There can be advantages to both methods but the structure search method has the important ability to be able to probabilistically find conditional independencies, rather then categorically allowing or denying an edge, based on a statistical test using an arbitrary confidence value. Also, with structure search, it is possible to find more than one high ranking network—this allows these various networks to be combined in a probabilistic model and produce better answers to queries. For these reasons, we will focus on the search method here.

Learning the structure of a Bayesian network is an NP-Hard problem and consequently enumeration and test of all network structures is not likely to succeed. In fact with just ten variables there are roughly 10^{18} possible DAGs, which leaves heuristic search methods through the space of different structures as possibly the only tractable solution.

In order to create a space in which to search through, three components are needed. Firstly all the possible solutions must be identified as the set of states in the space. Secondly a representation mechanism for each state is needed. Finally a set of operators must be given, in order to move from state to state in the space.

Traditionally, in searching for a Bayesian network structure, the set of states was the set of possible Bayesian network structures, the representation was a DAG and the set of operators were various small local changes to a DAG, e.g. adding, removing or reversing an arc. Successful application of the operators was also dependent on the changed graph being a DAG, i.e. that no cycle was formed in applying the operator. In keeping with the terminology used by Chickering this space shall be called B-space [2].

Once the search space has been defined, two other pieces are needed to complete the search algorithm, a scoring function which evaluates the "goodness of fit" of a structure with a set of data and a search procedure that decides which operator to apply, normally using the scoring function to see how good a particular operator application might be. An example search procedure is greedy search, that at every stage applies the operator that produces the best change in the structure, according to the scoring function. As for the scoring function, various formulæ have been found to see how well a DAG fits a data sample, e.g. by giving the relative posterior probability [8], or using a large-sample approximation such as the Bayesian information criterion [7].

3 Searching in the Space of Equivalence Classes

According to many scoring criterion, there are DAGs that are equivalent to one another, in the sense that they will produce the same score as each other. Looking at this in more depth, it is found that these

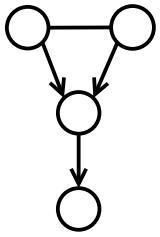


Figure 2: An example of a PDAG

equivalent DAGs produce the same set of independence constraints as each other, even though the structures are different. Independence constraints show how a set of variables are influenced or dependent on another set of variables, given a certain third set of variables. These constraints can be checked by analysing the DAG for certain structures. It turns out according to a theorem by Verma and Pearl that two DAGs are equivalent iff they have the same skeletons and the same v-structures [12]. By skeleton is meant the undirected graph that results in undirecting all edges in a DAG and by v-structure (sometimes referred to as a morality) is meant a head-to-head meeting of two arcs, where the tails of the arcs are not joined. From this notion of equivalence, a class of DAGs that are equivalent to each other can be defined, notated here as $Class(\mathcal{G})$.

Because of this apparent redundancy in the space of DAGs, attempts have been made to conduct the search for Bayesian network structures in the space of equivalence classes of DAGs [2, 3, 9]. The search set of this space is the set of equivalence classes of DAGs. To represent the states of this set, a different type of structure is used, known as a partially directed acyclic graph (PDAG). A PDAG (an example of which is shown in Figure 2)is a graph that contains both undirected and directed edges and that contains no directed cycles and will be notated herein as $\mathcal{P}$. Again, the equivalence class of DAGs corresponding to a PDAG is denoted as $Class(\mathcal{P})$, with a DAG $\mathcal{G} \in Class(\mathcal{P})$ iff $\mathcal{G}$ and $\mathcal{P}$ have the same skeleton and same set of v-structures. Related to this is the idea of a *consistent extension*. If a DAG $\mathcal{G}$ has the same skeleton and the same set of directed edges as a PDAG $\mathcal{P}$ then it is said that $\mathcal{G}$ is a consistent extension of $\mathcal{P}$. Not all PDAGs have a DAG that is a consistent extension of itself. If a consistent extension exists, then it is said that the PDAG *admits* a consistent extension. Only PDAGs that admit a consistent extension can be used to represent an equivalence class of DAGs and hence a Bayesian network.

It turns out that directed edges in a PDAG can be either compelled, or made to be directed that way, whilst others are reversible, in that they could be undirected and the PDAG would still represent the same equivalence class. From this idea, we can define a completed PDAG (CPDAG), where every undirected edge is reversible in the equivalence class and every directed edge is compelled in the equivalence class. We shall denote a CPDAG as $\mathcal{P}^{\mathcal{C}}$. It can be shown that there is a one-to-one mapping between a CPDAG $\mathcal{P}^{\mathcal{C}}$ and $Class(\mathcal{P}^{\mathcal{C}})$.

With this representation of equivalence classes of Bayesian network structures and a set of operators that modify the PDAGs which represent them (e.g. Insert an undirected arc, Insert a directed arc etc.), a search procedure can proceed. But one might ask why go to the bother of this type of search. Firstly, an equivalence class can represent many different DAGs in a single structure. Search in the space of DAGs often moves between states with the same equivalence class and so in a sense is wasted effort. This also affects the connectivity of the search space, in that the ability to move to a particular neighbouring equivalence class can be constrained by the particular representation given by a DAG.

There is also the problem given by the prior probability used in the scoring function in that whilst searching through the space of DAGs, certain equivalence classes can be over represented by this prior because there are many more DAGs contained in the class.

These concerns have motivated researchers with the results that recent implementations of algorithms that search through the space of equivalence classes have produced results that show a marked improvement in execution time and a small improvement in learning accuracy, depending on the type of data set [3, 4].

3.1 Techniques for Searching through Equivalence Classes

Note that here we refer to a *move* as an application of an operator to a particular state in the search space.

To be able to conduct a search through the space of equivalence classes, a method must be able to find out whether a particular move is valid and if valid, how good that move is. These tasks are relatively easy whilst searching through the space of DAGs—a check whether a move is valid is equivalent to a check whether a move keeps a DAG acyclic. The goodness of such a move is found out by using the scoring function, but rather then scoring each neighbouring DAG in the search space, the decomposability of most scoring criterion can be taken advantage of, with the result that only nodes whose parent sets have changed need to be scored.

However, this task of checking move validity and

move score is not as easy in the space of equivalence classes. For one, instead of just checking for cycles, checks also have to be made so that unintended v-structures are not created in a consistent extension of a PDAG. Scoring a move also creates difficulties, as it is hard to know what extension and hence what changes in parent sets of nodes will occur, without actually performing this extension. Also, a local change in a PDAG *might* make a non-local change in a corresponding extension and so force unnecessary applications of the score function.

These problems were voiced as concerns by Chickering [2]. In this paper he performs validity checking of moves by trying to obtain a consistent extension of the resulting PDAG—if none exists then the move is not valid. Scoring the move was achieved by scoring the changed nodes in the consistent extension given. These methods were very generic, but resulted in a significant slowdown in algorithm execution, compared to search in the space of DAGs.

To alleviate this problem, authors proposed improvements that would allow move validity and move score to be computed without needing to obtain a consistent extension of the PDAG [10, 9, 3]. This was done by defining an explicit set of operators, with each operator having a validity test and corresponding score change function, that could be calculated on the PDAG. These changes resulted in a speedup of the execution time of the algorithm, with the result that search in the space of equivalence classes of Bayesian networks became competitive with search in the space of Bayesian networks. However this improvement came at a price, in that the set of operators needed to be explicitly defined beforehand with corresponding validity tests.

4 Generically Scoring a Move in the State Space of Equivalence Classes of DAGs

The problem stated at the end of the last section leads onto the main result in this paper, a method to score a move in the space of equivalence classes of Bayesian networks, without having to have an operator and corresponding scoring function explicitly defined.

The method draws inspiration from the algorithm of Dor and Tarsi in finding consistent extensions of PDAGs and from the work of Chickering in searching through the space of equivalence classes of Bayesian networks [6, 3]. The main part of the search is like other similar search algorithms, in that it tries to find out which moves are valid and the scores for these valid moves. It then provides this information to a function which decides which move to take. Where the method differs is when it tries to score a particular move. A move in this sense is a set of primitive moves taken from a finite list shown in Table 1. It can be seen that composition of these operators is enough to represent any arbitrary change in a PDAG.

Before explaining the operation of the method, some notation will be explained. $\mathcal{P}_b(X)$, where X is a set of nodes refers to the subgraph of $\mathcal{P}_b$ restricted to the nodes in X. M refers to a move composed of a set of primitive moves. M_{XX} refers to the set of nodes in M, restricted to a particular type of primitive move, as given in Table 1. M^x refers to the set of all the x nodes in the move M. $M^{(x,y)}$ refers to the set of all pairs of nodes from each move in M. $Pa(x)$ stands for the parent set of x, whilst N_x stands for the set of neighbours of x, i.e. the set of node connected to x by undirected arcs. Finally $\Omega_{x,y}$, stands for $Pa(x) \cap N_y$ and $\Omega N_{x,y}$ stands for $(Pa(x) \cup N_x) \cap N_y$. The implementation of the method is shown in the algorithms ORIENT-ARCS and SCORE-MOVE. The ORIENT-ARCS algorithm is similar to the EXTEND algorithm given by Dor and Tarsi, in that it tries to find a consistent extension of a PDAG. In this case however, the extension is limited to a set of nodes given by the variables V, V^{pref} and O. Also at each iteration, tests and operations are performed on two PDAGs, $\mathcal{P}_b$ and $\mathcal{P}_a$, being representations of the PDAG before and after the application of a valid move. The fact that undirected arcs are directed in the same direction in both $\mathcal{P}_b$ and $\mathcal{P}_a$, helps ensure that the least number of changes are made in the local structure of the extension. The reason that there are three sets of nodes as opposed to the one set in EXTEND, is that the different sets correspond to different types of node in the PDAG. The set O consists of nodes which are not participating in operations on undirected arcs, but nevertheless can influence the parent set of the nodes that do participate. For this reason, nodes in O are the first to be directed away, if possible, as this reduces the chances of a change occurring in one of the participating nodes.

The next set V^{pref} consists of nodes that will be picked to direct away in preference to nodes in V, if possible. This is because certain combinations of moves can by scored on less nodes than normal, if the participating nodes are merged together into V^{pref}. Finally, the set V consists of nodes that participate in undirected operations and are not included in V^{pref}.

The SCORE-MOVE algorithm is the main algorithm that uses ORIENT-ARCS as a subroutine. This algorithm works by first directing neighbouring arcs toward new heads in the operations that don't involve undirected arcs. It then calculates the sets of nodes V, V^{pref} and O, by examining all primitive

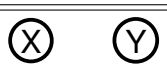
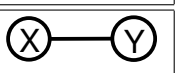
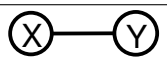
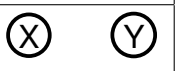
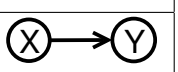
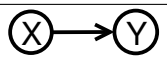
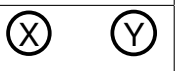
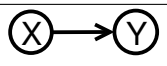
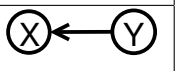
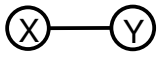
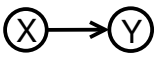
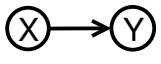
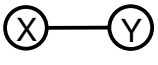
Operator	Set	Before	After
Insert Undirected	M_{IU}		
Delete Undirected	M_{DU}		
Insert Directed	M_{ID}		
Delete Directed	M_{DD}		
Reverse Directed	M_{RD}		
Direct Arc	M_{DA}		
Undirect Arc	M_{UA}		

Table 1: Primitive Operators

moves that involve undirected arcs. Next, it applies the move M on $\mathcal{P}_b$ to get $\mathcal{P}_a$. It then orients the arcs in $\mathcal{P}_b$ and $\mathcal{P}_a$, calculates which nodes have different parents sets in the PDAGs and from this calculates the score difference between the two PDAGs.

.

5 Proof of Validity

In this section the correctness of the various operations will be shown, starting with the SCORE-MOVE algorithm. The proofs developed by Chickering are used in many cases [3].

5.1 Correctness of SCORE-MOVE

The correctness of the first three statements in SCORE-MOVE can be inferred almost directly from Chickering. In this, necessary and sufficient conditions for the state of the PDAG are given for the *InsertD*, *DeleteD* and *ReverseD* operators. The only difference is that in SCORE-MOVE, $\Omega N_{x,y}$ is directed toward y as opposed to $\Omega_{x,y}$ in Chickering's proof. The difference between the former and latter sets is $N_{x,y}$. However, $N_{x,y}$ can be directed toward y without causing any cycles or v-structures. Therefore, the direction of $\Omega N_{x,y}$ toward y is sufficient. The reason for the difference, is that in Chickering's formulation, $Pa(x) \neq Pa(y)$ for an *InsertD* operation.

In defining the V^{pref} set, it can be seen that if $M_{DU}^{xy} \cap M_{IU}^{xy}$ contains elements, say e.g. z, then the undirected arcs corresponding to M_{DU}^{xy} and M_{IU}^{xy} can be directed toward z in $\mathcal{P}_b$ and $\mathcal{P}_a$, if possible and thus decrease the number of changed nodes. Likewise, if the undirected arcs in $\mathcal{P}_b$ are directed toward M_{DA}^{y}, this can decrease the number of changed nodes.

The V set is defined as containing all the nodes participating in an operation involving an undirected arc, less any nodes in the V^{pref} set.

The O set is defined as the intersection of all the neighbours of all the nodes in $V \cup V^{pref}$. Because of this, O contains all nodes connected by undirected arcs to both x and y for any operation involving an undirected arc between x and y.

The next two lines are simple function calls, whilst the line after defines the set of nodes whose parents have changed. It is obvious that any changes must include nodes from the sets M_{ID}^{y}, M_{DD}^{y}, M_{RD}^{xy} and any nodes described as changed by the ORIENT-ARCS algorithm. Finally, it is seen that the score difference of a compound move can be described as the score difference of each of the nodes whose parents sets have changed.

5.2 Correctness of ORIENT-ARCS

The purpose of ORIENT-ARCS is to direct arcs participating in moves involving undirected arcs, so that a consistent extension can be obtained of nodes involved in those moves. Nodes involved in these move are contained in V and V^{pref}, V^{pref} being used to aggregate certain moves together for efficiency purposes. However, in order to achieve this goal, nodes in the set $N_{x,y}$, where x and y and contained in $V \cup V^{pref}$, need to be taken account of, as these nodes cannot be directed away in both $\mathcal{P}_b$ and $\mathcal{P}_a$. These nodes are contained in set O.

In the running of ORIENT-ARCS, the idea is to select a node to direct all undirected arcs toward. This

Input: PDAG $\mathcal{P}_b$, PDAG $\mathcal{P}_a$, Set of nodes V, Set of nodes V^{pref}, Set of nodes O
Output: PDAG $\mathcal{P}_b$, PDAG $\mathcal{P}_a$, Set of nodes with new parents C

while $V \neq \emptyset || V^{pref} \neq \emptyset$ **do**
  $Nodes := V \cup V^{pref} \cup O$
  Let V_{min}, V_{min}^{pref} and O_{min} be sets such as follows
  $V_{min} = \{x | x \in V \wedge x$ has the least number of adjacencies in $\mathcal{P}_b(Nodes)$ and $\mathcal{P}_a(Nodes)$, with one adjacency being counted as zero adjacencies$\}$.
  V_{min}^{pref} and O_{min} are defined in similar ways
  if $O_{min} \neq \emptyset$ **then**
    $y \in O_{min}$
    $O := O \backslash y$
  else if $V_{min}^{pref} \neq \emptyset$ **then**
    $y \in V_{min}^{pref}$
    $V^{pref} := V^{pref} \backslash y$
    $C := C \cup y$
  else
    $y \in V_{min}$
    $V := V \backslash y$
    $C := C \cup y$
  end if
  Direct all undirected arcs toward y in $\mathcal{P}_b$ and $\mathcal{P}_a$
  $O := O \backslash \{z | z \in O \wedge A_z \equiv \emptyset$ in $\mathcal{P}_b(Nodes)$ and $\mathcal{P}_b(Nodes)\}$
  $V^{pref} := V^{pref} \backslash \{z | z \in V^{pref} \wedge A_z \equiv \emptyset$ in $\mathcal{P}_b(Nodes)$ and $\mathcal{P}_a(Nodes)\}$
  $V := V \backslash \{z | z \in V \wedge A_z \equiv \emptyset$ in $\mathcal{P}_b(Nodes)$ and $\mathcal{P}_a(Nodes)\}$
end while

Algorithm 1: orient-arcs

Input: PDAG $\mathcal{P}_b$, Valid compound move M
Output: Score Difference S

In $\mathcal{P}_b$, direct $\Omega N_{x,y}$ toward y, for all pairs (x,y) given by $M_{ID}^{(x,y)}$
In $\mathcal{P}_b$, direct N_y toward y, for all y given by M_{DD}^{y}
In $\mathcal{P}_b$, direct $\Omega_{y,x}$ toward x, for all pairs (x,y) given by $M_{RD}^{(x,y)}$
$V^{pref} := (M_{DU} \cap M_{IU}) \cup M_{DA}^{y}$
$V := (M_{IU} \cup M_{DU} \cup M_{DA} \cup M_{UA}) \backslash V^{pref}$
$O := \left(\bigcap N_{(V \cup V^{pref})} \right) \backslash \left(V \cup V^{pref} \right)$
$\mathcal{P}_a :=$apply move M on $\mathcal{P}_b$
$(\mathcal{P}_b, \mathcal{P}_a, C_u) :=$ orient-arcs$(\mathcal{P}_b, \mathcal{P}_a, V, V^{pref}, O)$
$C := C_u \cup M_{ID}^{y} \cup M_{DD}^{y} \cup M_{RD}$
$S := \sum_{c \in C} \left(\text{score}_{\mathcal{P}_b}(c) - \text{score}_{\mathcal{P}_a}(c) \right)$

Algorithm 2: score-move

proceeds until arcs involving all nodes in $V \cup V^{pref}$, have been dealt with. In selecting a node, care must be taken that no additional v-structures are formed. By examining the proof given by Dor and Tarsi [6], it can be seen that if for a node x, A_x is a clique, where A_x are all nodes adjacent to x, then x is valid. However, if x is a member of the set of nodes such that it is adjacent to the least number of nodes in both $\mathcal{P}_b$ and $\mathcal{P}_a$, then A_x must be a clique, therefore x can be selected. If x is a member of $V \cup V^{pref}$, then obviously it is a node whose parent set has changed, so x is inserted into the set C. Finally, all nodes that are not adjacent to any other node in $\mathcal{P}_b$ and $\mathcal{P}_a$ can be deleted, as they cannot affect the computation in any way.

5.3 Complexity of Algorithm

Given that there are k primitive moves in the move M, there are at most $2k$ nodes participating in an undirected move. If there is a bound c on the number of neighbours N_x, then there will be at most $(2k+c)$ iterations in the loop of ORIENT-ARCS. Given that the number of adjacencies of a node can be stored alongside the graph information, the only operations that must be performed can be done in $(2k+c)$ time. Therefore the complexity of the ORIENT-ARCS algorithm is in $O\left((k+c)^2\right)$.

In SCORE-MOVE, all the other operations, apart from the scoring operations, can be performed in $O(c^2)$ time. Therefore, the total overhead for the algorithm compared to a direct scoring algorithm is in $O\left((k+c)^2\right)$.

6 Experimental Results

Some basic experiments were run, the results of which can be seen in Table 2. This shows the results of structure searches without a cache on the ALARM Bayesian network. Samples of various sizes were taken and run under the different conditions. Here Chickering 02 is a greedy search, using the operators defined in [3] and GES is the Greedy Equivalent Search algorithm defined in [4]. "Alg." stands for the scoring mechanism defined in the corresponding paper and "Frame." stands for the scoring mechanism that uses the generic scoring framework defined in this paper. As can be seen from the results, the framework only adds a small amount of extra time, of which the relative difference tends to decrease as the sample size increases.

	Chickering 02		GES	
Samples	Alg.	Frame.	Alg.	Frame.
500	263	327	347	425
1000	317	374	454	535
1500	470	535	502	613
2000	489	534	589	654

Table 2: Running times of algorithms in seconds

7 Conclusions

A method was introduced that enables arbitrary changes in a PDAG representing an equivalence class of Bayesian networks to be scored. It was shown that the overhead introduced by this method compared to directly evaluating changes by an explicit scoring operator was small. In exchange for this extra running time, the method offers a more general framework in which arbitrary changes can be scored, obviating the need for explicit scoring operators to be defined. This could encourage more rapid development of algorithms used to learn Bayesian network structures, bypassing the analysis and definition of a theory to support each explicit operator.

One obvious direction in which this work might proceed is to develop a method that checks the validity of a proposed change in a PDAG, whilst also calculating the score of an acceptable change, all whilst keeping algorithm complexity competitive with other methods.

Acknowledgments

The authors are grateful to Drs Stuart Aitken and Richard Jensen for their helpful discussions, whilst taking full responsibility for the views expressed in this paper.

References

[1] David M. Chickering. Learning Bayesian networks is NP-complete. In D. Fisher and H. Lenz, editors, *Learning from Data: Artificial Intelligence and Statistics V*, chapter 12, pages 121–130. Springer-Verlag, 1996.

[2] David M. Chickering. Learning equivalence classes of bayesian network structures. In Finn Jensen and Eric Horvitz, editors, *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 150–157, San Francisco, California, August 1996. Morgan Kaufmann.

[3] David M. Chickering. Learning equivalence classes of bayesian-network structures. *Journal of Machine Learning Research*, 2:445–498, February 2002.

[4] David Maxwell Chickering. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 3:507–554, November 2002.

[5] Paul Dagum and Michael Luby. Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial Intelligence*, 60(1):141–154, March 1993.

[6] Dorit Dor and Michael Tarsi. A simple algorithm to construct a consistent extension of a partially oriented graph. Technical Report R-185, Cognitive Systems Laboratory, Department of Computer Science, UCLA, 1992.

[7] David Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.

[8] David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, September 1995.

[9] Paul Munteanu and Mohamed Bendou. The EQ framework for learning equivalence classes of Bayesian networks. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 417–424, Washington, DC, USA, 2001. IEEE Computer Society.

[10] Paul Munteanu and Denis Cau. Efficient score-based learning of equivalence classes of bayesian networks. In Djamel A. Zighed, Henryk J. Komorowski, and Jan M. Zytkow, editors, *Proceedings of the 4th European Conference on the Principles of Data Mining and Knowledge Discovery, PKDD 2000*, volume 1910 of *Lecture Notes in Computer Science*, pages 96–105. Springer-Verlag, Heidelberg, 2000.

[11] Solomon Eyal Shimony. Finding maps for belief networks is NP-hard. *Artificial Intelligence*, 68(2):399–410, August 1994.

[12] Thomas Verma and Judea Pearl. Equivalence and synthesis of causal models. In Piero Bonissone, Max Henriona, Laveen Kanal, and John Lemmer, editors, *Proceedings of the 6th Annual Conference on Uncertainty in Artificial Intelligence*, pages 255–268, New York, 1991. Elsevier.